*In Proceedings of the International Workshop on Development of Intelligent Information Systems, Niagara-on-the-lake, Ontario, Canada, 1991; University of Toronto Press, (1991) pp. 42-46*

# Coping with Requirement Freedoms

Martin S. Feather[*] and Stephen Fickas[**]

Formal methods for software development employing formal specifications are being used to good effect in a number of real-world situations. Two key factors that impede the even more extensive application of these methods are the difficulty of manipulating formal specifications, and the difficulty of constructing specifications in the first place. Manipulation of formal specifications has motivated a good deal of research (verification, analysis and program transformation). Con-struction of formal specifications has, we believe, received less attention. Our focus is on this latter activity.

We argue that there is in fact a wide gap between the natural statement of requirements and a formal specification of the same. To understand this gap, we identify "freedoms" that requirements typically exhibit, but which specifications cannot tolerate (e.g., inconsistency, incompleteness). We also consider the processes that are applied to construct and use formal specifications, and compare the freedoms against the processes to determine the capabilities required of those processes to be successful.

## 1 Formal specifications and requirements

Formal specifications have been successfully used to help in software development by clarifying ideas, serving as the basis for developing programs, and providing feedback early in the development process. For a discussion of the utility of formal methods, formal specifications in particular, see [1]. It follows that the processes of construction, comprehension and modification of specifications (a necessary precursor to constructing, comprehending and modifying the software which is developed from, or compared with them) are vital to the success of this paradigm.

Much of the success of specification languages is attributable to their freedom from efficiency concerns. Specifications nevertheless impose a number of expectations, for example that they be complete, consistent and unambiguous. In contrast, the expressions of requirements will typically be fragmentary, contradictory, incomplete, inconsistent and ambiguous. Furthermore, their expression may employ widely varying levels of abstraction (e.g., concrete examples, scenarios, general properties) styles (e.g., textual, graphical, formulae, domain-specific notations) and viewpoints (e.g., system-wide properties, single user viewpoints, snapshots of the entire state at one moment in time, historical traces of ongoing activity).

We term these properties "requirement freedoms", to emphasize their importance for facilitating the acquisition, comprehension and validation of requirements. They are not necessarily failings or errors of the producer of those requirements; in fact, it is often advantageous to make use of these freedoms. Because they are the antithesis of desired properties of formal specifications,

[*] This author has been supported in part by Defense Advanced Research Projects Agency grant No. NCC-2-520, and in part by Rome Air Development Center contract No. F30602-85-C-0221 and F30602-89-C-OI03. Views and conclusions contained in this document are those of the author and should not be interpreted as representing the official opinion or policy of DARPA, RADC, NSF, the U.S. Government, or any other person or agency connected with him.
[**] This author is currently at the Computer Science Department at the University of Oregon, and has been supported in part by NSF grant CCR-8804085.

however, the task of developing a formal specification from such requirements (and ultimately a program, whether or not this is by route of a specification) may be complex and lengthy.

We feel that there is considerable scope for theories, techniques and tools to assist in bridging this gap between requirements and specifications, and to make use of requirements before they have been completely turned into specifications. Such tools must not only be tolerant of requirement freedoms, but must actively assist in the removal of these freedoms in order to produce specifications.

## 2 Requirement freedoms

We list some properties that requirements may, indeed typically do, exhibit, yet are generally not acceptable for formal specifications.

**Incompleteness** Requirements can provide partial descriptions of tasks. Particularly for large and complex tasks, it is important to be able to provide requirements incrementally, so that at any intermediate point there they may be incomplete. It is also useful for several people to be separately preparing or extending sets of requirements, which later may be combined into a single whole; for maximum advantage, each of those people should not be constrained to be working with a complete set of requirements.

**Inconsistency** Requirements can be mutually contradictory. This is particularly useful for the statement of an idealized requirement, which we may recognize as being unrealizable, but which we nevertheless wish to express. This is also useful for dealing with contradictory viewpoints, in order to describe them as a precursor to negotiation. Finally, multiple sources of requirements need not be tightly coupled so as to constrain them to generating only completely consistent sets of requirements.

**Redundancy** Requirements need not avoid stating the same thing multiple times. Indeed, it may be advantageous to be able to state the same thing from multiple points of view, e.g., the cash register's point of a transaction view as compared with the customer's point of view of a transaction. Both viewpoints may overlap, stating the same thing in different ways. Furthermore, redundancy permits cross-checking, a form of validation of the accuracy of the emerging specification.

**Ambiguity** Requirements may be ambiguous. In particular, holding to an attitude of “all or nothing” is often counter-productive during the requirements process. This can be addressed in several ways. A client should be allowed to state requirements in an ideal fashion, but at the same time list acceptable alternatives. If it is difficult to know alternatives ahead of time, a client should be allowed to state flexible requirements (e.g., try hard to meet this goal) or preferences of one requirement over another (e.g., always choose safety over cost if both can't be met).

**Non-uniformity of abstraction** Requirements may express knowledge anywhere within a broad spectrum of generality. This range goes from very general-purpose domain independent, through domain-specific, task-specific, to concrete examples.

**Heterogeneous forms of expression** Statements of requirements should be free to use what-ever form of expression is most natural to the aspect of the task being described. This might be textual, formulae, graphical, a domain-specific notation, etc. Furthermore, differ-ent aspects of the same task should be able to utilize different forms of expression. This is convenient for both human understanding (presentation in those terms that individual is most comfortable with), and reasoning (expression that suppress details irrelevant to the particular form of reasoning).

We term these requirement freedoms, based on their utility at the time of stating and rea-soning about requirements, but their undesirability as properties of specifications (or indeed of implementations derived from such specifications). This is analogous with the notions of specifi-cation freedoms as being properties of specifications (e.g., the freedom to state what the result of some computation must be without being required to state how to do that computation) that are conducive toward clear, concise and comprehensible specifications, but which must be removed to achieve efficient implementations of those specifications [2].

## 3 Processes on requirements and specifications

We now turn our attention to the processes that operate on requirements and specifications, and then consider how these processes interact with the requirement freedoms we have identified.

### *3.1 Acquisition and Construction*

Requirements knowledge may be gathered from a number of sources. Clients provide sets of requirements, a body of domain knowledge may already be available, and case histories may have been kept. The appropriate pieces of knowledge must be acquired and combined, and irrelevant knowledge discarded.

### *3.2 Explanation and Analysis*

To be comprehensible, descriptions of requirements and specifications must take a number of forms, and be suitably structured. The viewer's familiarity with various notations, and the suitability of notations to portraying the particular information, will dictate the choice of presentation. Examples of structuring include outlines, examples/cases, summaries (including qualitative models), indexes, scenarios, and several points of view.

### *3.3 Modification*

Requirements will frequently need changing during specification construction as the various re-quirement freedoms are removed. For instance, inconsistencies among requirements will lead to their revision. Attempts to specify and implement ideal requirements may show them to be unre-alizable or just too costly, thus leading to their weakening. Modifications to requirements will also occur in response to external changes (e.g., an extension of the behavior desired of the system being constructed, a change to the environment within which that system resides, the desire to build a similar but not identical system). Modification also permits a form of “near-miss” description, in which something close to the desired requirement is easily described (perhaps by merely selecting it from a number of representative examples), and then tailored to form the desired requirement. This tailoring is a form of modification.

## 4 Requirement freedoms and processes

We do not want to wait until we have formed a specification before we can begin to apply our methods and tools. Thus these methods and tools must be tolerant of requirements freedoms. Lack of such tolerance would force the user to have manipulated the requirements ahead of time as a prerequisite to expression them. This capacity for tolerance necessitates careful design of tools so that they continue to function reasonably well in the face of these freedoms, both in terms of preventing nonsensical effects, and in terms of not overly disrupting the process with numerous queries or warnings to the user.

Furthermore, we want to use tools to help bridge the gap between requirements and specifications. We examine each of our requirement freedoms to see what support is needed to deal with their effect on these specification processes:

Incompleteness During construction, incompleteness may direct the acquisition of further re-quirements information, or modification of already acquired requirements. The information that is imparted in an incomplete set of requirements must be retained and manipulated, even if it remains disjointed and cannot be coalesced into a single whole. Analysis and ex-planation tools must handle incompleteness gracefully. They should do reasoning about, and presentation of, the information that is available, while indicating the limits. Modification of requirements may introduce further incompleteness, or be applied to remove it. Tools should make plausible guesses when appropriate, and be able to gracefully retract those guesses when they are determined to be incorrect.

Inconsistency identifies conflicts in the requirements that require resolution. This is a frequent activity in requirements analysis, where trade-offs must be made between conflicting ideals. Negotiation techniques will be brought to bear here. Analysis must appropriately limit the propagation of inconsistency so that an inconsistency amongst the requirements does not render all other analysis results worthless!

Redundancy permits the cross-checking of information. A single formalization of the information will be selected for inclusion in the specification, but linked back to the multiple requirements that led to it. Analysis and explanation techniques can make good use of redundancy when presenting information to the user, since alternative presentations of the same information may emphasize different aspects that would otherwise be easily overlooked by the user. In-deed, analysis is the activity of presenting redundant information, although presumably in-formation that would be hard to immediately deduce from a given form of the specification (e.g., proving some property holds of a specification is 'merely' a deduction from the existing specification, yet arbitrarily complex reasoning may be required to make that deduction).

Ambiguity indicates the need for choice among alternatives. Our tools must be able to navigate in a sea of choices, preferences and trade-offs. It may no longer be possible to say that a requirement can or cannot be met. Instead, achievement becomes relative to a client's preferences or to some allowable range of behavior. Further, during construction there should be no pre-ordained dogma that insists choices be resolved when they arise. It is useful to retain a record of alternatives, and justifications for choices when they are made, so that the resulting system can be justified, and so that future changes can induce a reconsideration of those choices as appropriate.

Non-uniformity of abstraction allows the use of general-purpose information suitably instan-tiated for the task in hand. Similarly, specific but paradigmatic behaviors may be stated as requirements, the intent being to construct a specification that exhibits those, and similar, behaviors; thus generalization from examples occurs during construction. Analysis and ex-planation likewise make use of this flexibility, presenting information in general terms (e.g., a traffic light controller is an instance of a scheduling system), very specific cases that the user can relate to (e.g., car-1 enters the intersection while its traffic light is green), and even in qualitative terms (e.g., an increase in traffic will cause a decrease in a throughput require-ment). Modification comes in to play not only to move between levels of abstraction in the traditional sense of refinement/generalization but also as a means to tailor, customize and incrementally construct something 'similar' without being restricted to pure refinement or abstraction.

Heterogeneous forms of expression must be acceptable as input forms of requirements, and require translation into some common (internal) representation. Support for user-defined languages and notations permit task-and domain-specific extension of the notations available. The external, user-provided expressions should be used where possible as the means to present analysis information, so that translation must go both ways. Modifications expressed in terms of any of the representations must be appropriately conducted in the internal representation.

The concept of requirement freedoms, and the language and tools they engender, are being studied as part of the Aries project at ISI and Lockheed Sanders, and the Kate project at Oregon. Further information may be obtained by contacting the authors: feather@isi.edu, fickas@cs.uoregon.edu.

## References

[1] A. Hall. Seven myths of formal methods. IEEE Software, 7(5):11-19, September 1990.

[2] P.E. London and M.S. Feather. Implementing specification freedoms. In C. Rich and R. Waters., editors, Readings in Artificial Intelligence and Software Engineering, pages 285-305. Morgan Kaufmann, 1986. Originally published in Science of Computer Programming, 1982 No.2, pp 91-131.